GENIE Automated Validation Suite

Gabriel N. Perdue
Fermilab
2014/September/11

GENIE

- Generates Events for Neutrino Interaction Experiments.
- http://genie.hepforge.org
- Well-engineered C++ software framework built on sound OO-principles and design patterns. (The Gang of Four is omnipresent.)
- Propagates a flux of neutrinos (specified by function, histogram, or ntuple) through a geometry (Geant4-compatible) and simulates the initial interaction and propagation of hard vertex products through the nuclear medium. Geant4 takes over when particles leave the nucleus.
- ROOT provides many core utilities. GENIE also heavily leverages other HEP and FOS software - LHAPDF, GSL, Pythia, log4cpp, etc.

Andreopoulos, C. and Bell, A. and Bhattacharya, D. and Cavanna, F. and Dobson, J. and others. "The GENIE Neutrino Monte Carlo Generator". Nucl.Instrum.Meth. A614. 87-104. 2010.

GENIE at FNAL

- GENIE is the primary event generator for:
  - ArgoNeut
  - LAr1-ND
  - LBNE
  - MicroBooNE
  - MINERvA
  - NOvA
- GENIE is also being considered for special studies by MINOS and MiniBooNE (they use previous generation software for their main generators).

Software Dependencies

- GENIE uses ROOT (5, eventually 6), Pythia(6, eventually 8), LHAPDF (5, eventually 6), log4cpp, GSL.
- Libraries: libstdc++, libc, libgcc, linux-vdso, libm, ld-linux-x86-64, libxml2 ... possibly not complete (ROOT, etc. have requirements).
- GENIE does not (yet) use any features from C++11.
- Generally, building on Scientific Linux is easy.
- Building 32-bit is also possible.

```
> ./cloc-1.60.pl R-2_8_0/
    3285 text files.
    3200 unique files.
    7197 files ignored.

http://cloc.sourceforge.net v 1.60  T=113.14 s (11.3 files/s, 4119.1 lines/s)
--------------------------------------------------------------------------------
Language                      files          blank        comment           code
--------------------------------------------------------------------------------
C++                             525          30478          37587         176349
XML                             125          21895           2144         147176
C/C++ Header                    504           9052           8118          22282
Perl                             28            456           1469           3620
make                             47            514            485           1651
Bourne Shell                     34            157            334           1059
Bourne Again Shell                2            145            127            727
SQL                              12             37              0            117
--------------------------------------------------------------------------------
SUM:                           1277          62734          50264         352981
--------------------------------------------------------------------------------
```

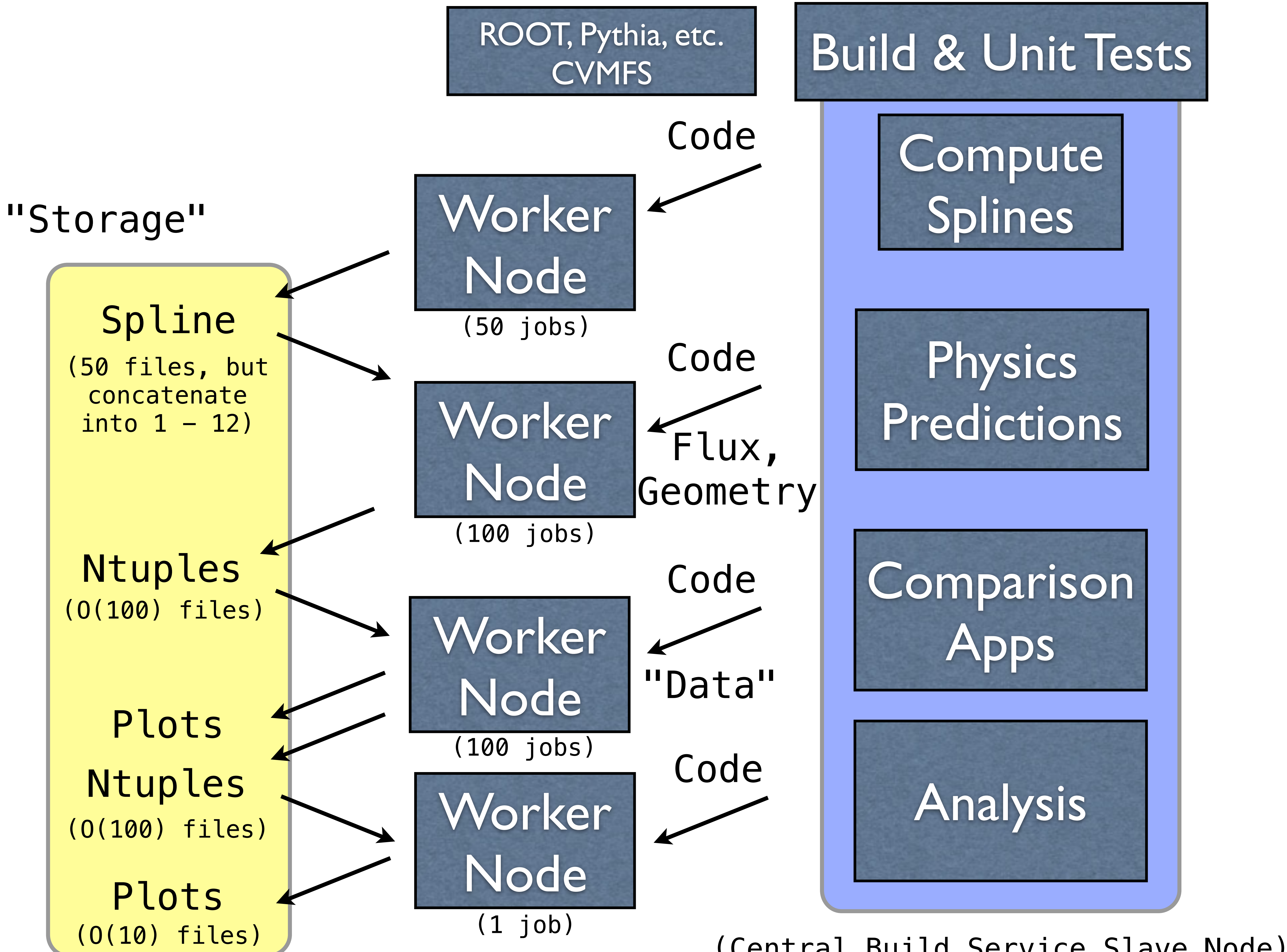
There is a lot of configuration XML and experimental data packaged for the validation framework.

Basic Goals

- The goal is a "DC operation" with occasional spikes around the time of a new release (twice annually):
  - Nightly simple tests (build, unit tests)
  - Weekly / Nightly (eventually... maybe faster) integration branch full physics validation test
  - The validation will grow in physics complexity over time, but operational complexity should be ~flat.
    - Each new validation app will have similar inputs/ outputs and interfaces.
- Push-button physics validations of the development branch
- Global tuning (model variation, physics validation, fit for optimal parameters) may require the OSG.

Scripting Framework

- Six Stages:
  - Build
  - Unit Tests (Eventually)
  - Generate Cross Sections (In: NA; Out: XML)
  - Generate Physics Predictions (In: XML, Geometry, Flux; Out: ROOT Ntuples)
  - Run Data/MC Comparison Apps (In ROOT Ntuples; Out ROOT Ntuples, PDFs)
  - Compare outputs to previous data/MC comparisons / Study global behavior (In ROOT Ntuples; Out ROOT Ntuples, PDFs)
- Each step depends on the previous step succeeding.
- We plan on using the Central Build Service to coordinate the flow from one stage to the next, but each stage will have its own script.



ROOT, Pythia, etc.
CVMFS
Build & Unit Tests
Compute Splines
Physics Predictions
Comparison Apps
Analysis
(Central Build Service Slave Node)
Code
Code
Flux, Geometry
Code
"Data"
Code
Worker Node
(50 jobs)
Worker Node
(100 jobs)
Worker Node
(100 jobs)
Worker Node
(1 job)
"Storage"
Spline
(50 files, but concatenate into 1 - 12)
Ntuples
(O(100) files)
Plots
Ntuples
(O(100) files)
Plots
(O(10) files)

Build

- Script in hand that does green-field builds with minimal checks for previously installed 3rd party codes.
- Currently assumes x86_64.
- Not CVMFS aware.
- Bash. Willing to rewrite into Python – it is missing some functionality anyway, e.g., how to declare files to SAM, etc.
    - Requires Git, wget, gcc 4.1+ (can probably go lower), gfortran, Python (2.6?) for the full stack (ROOT + Pythia6 + LHAPDF5).

Generate Cross Sections

- "Bootstrap Option": Keep a set of splines in /pnfs/data(?) for testing other components of the scripting framework.
- Comparison: Store a complete set of splines from official releases, the last X validation releases (where X is probably one for automated comparisons).
- ~20–50 grid jobs if we do one job per target (could choose to break up the Event Generator Lists).
- Output at this stage is ~20–50 x ~20 MB files, which could be concatenated into a smaller set of files (or one file).
  - One for free nucleons, one for each element (favorites: He, C, O, N, Ar, Fe, etc. Minerva has ~20.).
- Jobs in the next stage must load the file, and the inefficiency in loading unneeded cross sections is outweighed by simpler coordination.

Generate Physics Predictions

- Each prediction is a unique snowflake.
  - Like snowflakes though, very topologically similar in terms of I/O requirements, etc.
- Requires the cross section spline and support files: a flux and target specification (likely to complicated, but small):
  - e.g., the NuMI flux on the MINERvA geometry
  - Sometimes very simple, e.g. 500 MeV electrons on carbon

Comparison Applications

- Each application is a unique snowflake.
  - Similar topologies...
- Requires a published data set and the generator predictions file.
- Jobs should be very fast (unless they are doing a complicated fit, etc.).
- Output is plots and ROOT files (histograms and/or ntuples).

Analysis Stage

- Vaporware.
- In principle, not difficult to write a very basic placeholder that only does a few simple things.
- Use the placeholder to design the I/O and workflow, then add features using only "physicist time."